IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application	)	<u>PATENT APPLICATION</u>
	)	
Inventor: Cirne et al.	)	Art Unit: 2192
	)	
Application No.: 10/700,338	)	Examiner: Wei, Z.
	)	
Filed Date: November 3, 2003	)	
	)	
Title: SIMPLE METHOD OPTIMIZATION	)	
	)	<u>Customer No. 105234</u>
	)	

<u>APPEAL BRIEF</u>

Mail Stop Appeal Brief – Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This brief is submitted in accordance with 37 C.F.R. §41.37, following the Notice of Appeal filed by Appellant(s) on June 5, 2012. The fee set forth in 41.20(b)(2) is submitted herewith.

Table of Contents

I. REAL PARTY IN INTEREST *(37 C.F.R. §41.37(c)(1)(i))*

The real party in interest is CA, Inc.

II. RELATED APPEALS AND INTERFERENCES *(37 C.F.R. §41.37 (c)(1)(ii))*

Appellant knows of no other appeals or interferences which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

III.   SUMMARY OF CLAIMED SUBJECT MATTER *(37 C.F.R. §41.37(c)(1)(iii))*

Claims 1-2, 5-13, 17-23, 28-30, 32-35, 37-42, 44-47, 51 and 52 are pending in this application.   Claims 39 and 47 are allowed. Claims 5, 17, 23, 37, 41 and 51 are objected to as being dependent upon rejected base claims, but would be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims.

Claims 1-2, 6-13, 18-22, 28-30, 32-35, 38, 40, 42, 44-46, and 52 stand finally rejected. Claims 3-4, 14-16, 24-27, 31, 36, 43, 48-50 and 53-59 were cancelled previously, without prejudice.

Appellant herein appeals from the final rejection of claims 1-2, 6-13, 18-22, 28-30, 32-35, 38, 40, 42, 44-46, and 52.

Overview

The technology recited in claims 1-2, 6-13, 18-22, 28-30, 32-35, 38, 40, 42, 44-46, and 52 generally relate to "technology for monitoring applications" (Specification p. 2, lines 16-17). As discussed in the Background of the Invention section, prior monitoring techniques, such as the implementation of a performance analysis tool, provide an indication of an application's performance.   These indications may, for example, "provide timing data on how long each method (or procedure or other process) is being executed, report how many times each method is executed and/or identify the function call architecture" (Specification p. 1, lines 22-25). However, many times these tools provide too much data, which problematically creates for the user the "difficult task of analyzing the multitude of data to determine which data is relevant and which data is not relevant (e.g. redundant or otherwise not useful)" (Specification p. 2, lines 3-4). In many instances, prior monitoring techniques would "instrument a large number of methods in the software in order to be able to analyze the performance of each method.   However, modifying a large number of methods... [adds] an enormous amount of code to the software and may impact performance of the underlying software" (Specification p. 2, lines 11-13).

The technology recited in the claims for the present application resolves these issues by

-5-

only modifying certain methods for purposes such as monitoring or tracing, making the process of monitoring or tracing applications much more efficient and useful. In one embodiment, methods that are determined to be complex are modified. Methods that are determined to be simple are not modified. As stated in the Specification:

> In one embodiment, a method is complex if it meets three criteria: (1) the method has an access level of public or package; (2) the method is non-synthetic and (3) the method calls at least one other method. Methods that do not satisfy all three criteria are classified as simple methods. In other embodiments, a method can be classified as complex if it satisfies two of the above criteria, or other similar criteria (p. 12, lines 17-23).

Software typically contains methods that call other methods. For example, in Figure 4 of the Drawings, method M1 calls methods M2 and M3, and method M3 calls methods M4 and M5, both of which do not call another method (also see Specification p. 11, line 23 – p. 12, line 11). Monitoring M1 would result in the monitoring of all of the methods that it calls. It would be inefficient and redundant to also monitor the methods that it calls separately. Therefore, it is useful to monitor only methods which call at least one other method.

The Specification describes a method's access level as follows:

> Java provides for four levels of access control for methods: public, private, protected, and package. Private is the most restrictive access level. A private method can only be accessed by methods in the same class. A protected method can be accessed by other methods in the same class, sub classes and classes in the same package. A public method can be accessed by any class of any parentage in any package. The package access level is the default and allows a method to be accessed by classes in the same package, regardless of their parentage" (p. 12, line 24 – p. 13, line 5).

In the example code found on page 13, line 13 to page 14, line 9 of the Specification, the access level can be determined by referencing the access level indicated prior to each method name (e.g. public methodOne()).

A method is determined to be synthetic if it is "a method that does not appear in the source code. For example, during compilation, the compiler may add one or more methods.

<center>-6-</center>

Typically, the compiler explicitly makes it easy to see that methods are or are not synthetic. Java compilers flag these methods with the 'Synthetic' attribute" (Specification p. 13, lines 6-11).

As shown in Figure 5 of the Drawings, "various methods are accessed and determined to be either complex or simple... If the method is complex, it is modified... If the method is determined to be simple, it is not modified for the purposes that the complex methods are modified" (Specification p. 14, lines 19-23). Figure 6 of the Drawings shows one way a method that is determined to be complex can be modified. A method is modified by accessing the beginning of the byte code for the method (step 302), adjusting the byte code indices to prepare for the addition of code (step 304), adding new start byte code (step 306), accessing the end of the byte code (step 308), adding new exit byte code (step 310), adjusting the exception table due to the adjustment of the byte code indices (step 312), and adding a new exception table entry in the exception table (step 314) (also see Specification p. 15, line 14 – p. 16, line 21). Such modification to the code for a method can be performed for any purpose, such as monitoring or tracing the method.

Summary of Independent Claim 1

The method recited in independent claim 1 relates to a "process for monitoring." Figure 3 of the Drawings describes a process for monitoring using a modified method, which includes receiving existing code (step 260), receiving new function(s) such as "new classes and methods that allow for monitoring the application" (step 262), modifying existing code (as described in Figure 6, for example) (step 264), adding "all or part of the new functionality (e.g. the new classes/methods)... [to] the existing code" (step 266), storing the modified code (step 268), and running the modified code for monitoring (step 270) (Specification p. 10, lines 10-21). The Specification describes that the process for monitoring may be implemented, for example, through "software that is stored on one or more processor readable storage devices [that] is used to program one or more processors (p. 10, lines 1-2).

Claim 1 further recites, "accessing a method." One example of this feature is shown in step 280 of Figure 5 of the Drawings. The "method is accessed from the set of methods that are to be considered" for monitoring (Specification p. 14, lines 24-25).

-7-

Claim 1 further recites "automatically determining whether to modify said method, said step of automatically determining whether to modify said method includes automatically determining whether said method calls another method and whether said method has an access level that satisfies a criterion." One example of these features is shown in Figure 5 of the Drawings in the decision block of step 286. "If the method under consideration does not call any other methods, then that method is not modified for the purposes that the complex methods are modified (step 290). If the method under consideration does call one or more methods, then the method under consideration is modified" (Specification p. 15, lines 6-8). A further example is shown in step 282 of FIG. 5. In step 282, it is determined whether the method has an access level of public or package access level. If the method does not have an access level of public or package, then that method is not modified for the purposes that the complex methods are modified (Specification p. 14, line 26 – p. 15, line 2).

Claim 1 further recites "modifying said method for a particular purpose only if said method calls another method and said access level satisfies said criterion." As stated above and shown in Figure 5 of the Drawings, the method is modified if the method calls other methods and said access level satisfies said criterion (step 288). The method can be modified for a particular purpose, such as "monitoring [an] application" (Specification p. 10, line 13), for example. The monitoring for a particular purpose may include the addition of "functionality in order to reduce overhead and reduce the amount of data generated, without losing important data" (Specification p. 10, lines 23-24).

Summary of Independent Claim 13

The method recited in claim 13 also relates to a "process for monitoring." The process for monitoring is carried out by "automatically determining which methods of a set of methods call one or more other methods and are synthetic." This step of automatically determining is described and shown in Figure 5, steps 284 and 286 and Specification p. 15, lines 2-13. "[I]n step 284 it is determined whether the method is synthetic. If the method is synthetic, then that method is not modified for the purposes that the complex methods are modified (step 290). If the method is not synthetic, then in step 286 it is determined whether the method calls any other

-8-

methods. If the method under consideration does not call any other methods, then that method is not modified for the purposes that the complex methods are modified (step 290)."

Claim 13 further recites "using a first tracing mechanism for said methods that call one or more other methods and are not synthetic without using said first tracing mechanism for methods that do not call one or more other methods or are synthetic." Figure 5 of the Drawings shows that if the method is determined to not be synthetic (step 284) and to call other methods (step 286), the method will be modified (step 288) "to add the tracer" (Specification p. 15, line 9). "If the method under consideration does not call any other methods, then that method is not modified for the purposes that the complex methods are modified," such as to add the tracer (Specification p. 15, lines 6-9). The tracing mechanism is used when the modified code is run (step 270 of Figure 3 of the Drawings).

Summary of Independent Claim 22

The technology recited in claim 22 includes "one or more processor readable storage devices having processor readable code embodied on said processor readable storage devices, said processor readable code for programming one or more processors to perform a process." The Specification describes that the process for monitoring may be implemented through "software that is stored on one or more processor readable storage devices [that] is used to program one or more processors (p. 10, lines 1-2).

Claim 22 further recites "automatically determining which methods of a set of methods to modify, said step of determining includes automatically determining which methods call one or more other methods and have an access level of either public or package in the JAVA programming language." This step of determining is described and shown at least in Figure 5, steps 282 and 286 and Specification p. 9, line 24 – page 10, line 2; and page 10, lines 10-21. At page 12, line 24 – page 13, line 5, the Specification discusses, "an access level of either public or package in the JAVA programming language."

Claim 22 further recites "modifying for a particular purpose only those methods that call one or more other methods and have an access level of either public or package in the JAVA programming language." This step of modifying is described and shown at least in Figure 5, step

-9-

288 of the Drawings and on page 10 of the Specification.

Summary of Independent Claim 33

The technology recited in claim 33 includes "one or more processor readable storage devices," which has been discussed with respect to claim 22. Claim 33 further recites "automatically determining whether to trace a method, said step of determining includes automatically determining whether said method calls another method and if said method has an access level that satisfies a criterion." This feature is shown in Figure 5 of the Drawings in the decision block of steps 282 and 286 and Specification p. 14, line 26 – p. 15, line 2; and p. 10, lines 10-21.

Claim 33 further recites "tracing said method for a particular purpose only if said method calls another method and said access level satisfies the criterion. "If the method under consideration does call one or more other methods and said access level satisfies said criterion, then the method under consideration is modified to add a tracer" (Specification p. 15, lines 8-9). When the modified code is run, the method is traced (step 270 of Figure 3 of the Drawings).

Summary of Independent Claim 40

The apparatus recited in claim 40 includes "a storage device," such as the processor readable storage device disclosed in the Specification (p. 9, line 19). The claim further recites "one or more processors in communication with said storage device, said one or more processors perform a process." As discussed above, the apparatus can be, for example, a "computing devices... [with] one or more processors" (p. 9, line 18-19).

Claim 40 further recites "accessing a method" and "determining whether said method calls one or more different methods and can be called by a sufficient scope of one or more other methods." These features are discussed and shown in at least Figure 5, steps 282 and 286 and in the Specification at p. 14, line 26 – p. 15, line 2 and p. 10, lines 10-21. These features are also discussed at p. 13, line 17 – p. 13, line 5:

> There are many different standards that can be used within the spirit of the present invention to classify methods as simple or complex. In one embodiment,

-10-

a method is complex if it meets three criteria: (1) the method has an access level of public or package; (2) the method is non-synthetic and (3) the method calls at least one other method. Methods that do not satisfy all three criteria are classified as simple methods. In other embodiments, a method can be classified as complex if it satisfies two of the above criteria, or other similar criteria.

Java provides for four levels of access control for methods: public, private, protected, and package. Private is the most restrictive access level. A private method can only be accessed by methods in the same class. A protected method can be accessed by other methods in the same class, sub classes and classes in the same package. A public method can be accessed by any class of any parentage in any package. The package access level is the default and allows a method to be accessed by classes in the same package, regardless of their parentage.

Claim 40 further recites "tracing said method for a particular purpose only if said method calls one or more different methods and can be called by a sufficient scope of one or more other methods." These features are discussed and shown in Figure 5, step 288 and Specification p. 15, line 6-9.

Note that the explanations of the independent claims are not limited to those features referenced above by the Specification or Drawings. The features recited in the independent claims are disclosed throughout the Specification and Drawings, and the above descriptions merely serve as a concise explanation of the subject matter defined in each of the independent claims.

IV. ARGUMENT *(37 C.F.R. §41.37(c)(1)(iv))*

A) Rejection of Claims 1-3, 6-8, 10, 12-13, 18, 20, 40, 42, 44, and 52 Under 35 U.S.C. §103(a) as being unpatentable over *Berkley* in view of *Webster* and *Grove*

Applicants respectfully assert that the rejection of Claims 1-3, 6-8, 10, 12-13, 18, 20, 40, 42, 44 and 52 under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 6,351,843 to *Berkley et al.* (hereinafter "*Berkley*") in view of U.S. Patent No. 6,738,965 to *Webster* (hereinafter "*Webster*"), and in view of "Call Graph Construction in Object-Oriented Languages" to *Grove et al.* (hereinafter "*Grove*") is in error. The Applicants respectfully traverse the rejection for the following reasons.

Applicants will first address the Examiner's remarks on page 3 of the final Office Action ("final Office Action") mailed on March 9, 2012. Applicants respectfully assert that the Examiner has misquoted the Applicants' Specification by omitting selected portions of the text of the Instant Specification. In paragraph 6, the Specification states:

> *In one example*, a method is complex if it meets three criteria: (1) the method has an access level of public or package; (2) the method is non-synthetic and (3) the method calls at least one other method. Methods that do not satisfy all three criteria are classified as simple methods (emphasis added).

The Examiner has selectively omitted the phrase, "in one example" in the quote on page 3, lines 13-16 of the final Office Action. The Examiner has drawn the erroneous conclusion that, "it can reasonable (sic) interpreted that all the three criteria have to be determined and meet (sic) in order to modifying (sic) and monitoring (sic) the method/application."

Applicants respectfully assert that the above passage from paragraph 6 clearly is referring to one example. Therefore, determining whether a method is complex is not limited to determining whether a method has all three of these criteria.

Furthermore, the instant Specification clearly states in paragraph 35 that a method can be

-12-

classified as complex if it satisfies two criteria:

> There are many different standards that can be used within the spirit of the present invention to classify methods as simple or complex. In one embodiment, a method is complex if it meets three criteria: (1) the method has an access level of public or package; (2) the method is non-synthetic and (3) the method calls at least one other method. Methods that do not satisfy all three criteria are classified as simple methods. ***In other embodiments, a method can be classified as complex if it satisfies two of the above criteria***, or other similar criteria (emphasis added).

Therefore, the Specification clearly indicates that a method could be classified as complex if it satisfies two of the following criteria: 1) the method has an access level of public or package; (2) the method is non-synthetic and (3) the method calls at least one other method. Furthermore, the Specification clearly indicates that other similar criteria could be used to classify a method as complex.

i) Claim 1 is patentable under 35 U.S.C. §103(a) over *Berkley* in view of *Webster* and *Grove*

Claim 1 recites:

> A process for monitoring, comprising:
> accessing a method;
> automatically determining whether to modify said method, said step of automatically determining whether to modify said method includes automatically determining whether said method calls another method and if said method has an access level that satisfies a criterion; and
> *modifying said method for a particular purpose only if said method calls another method and said access level satisfies the criterion.*

Applicants respectfully assert that *Berkley* in view of *Webster* in further view of *Grove* fails to disclose, "*modifying said method for a particular purpose only if said method calls*

-13-

another method and said access level satisfies the criterion," as claimed.

Applicants respectfully assert that they have taught a fundamentally different technique than any combination of *Berkley*, *Webster*, and *Grove*. The solution recited in claim 1 does not to use brute force to determine a call graph (which may be inaccurate anyway). Instead, claim 1 recites an elegant solution that modifies a method for a particular purpose *only if said method calls another method and said access level satisfies the criterion*.

Note that claim 1 recites a solution that does not necessarily modify all methods that call another method. Note that a second condition must be met (*access level satisfies the criterion*). This prevents too many methods from being modified. Note that modifying too many methods can be inefficient. However, determining which methods to modify can be difficult. Applicants' solution provides an efficient way to determine which methods to modify, while not modifying too many methods.

Applicants have carefully reviewed the rejection of claim 1 laid out from pages 6 – 8 of the Office Action and do not see where the claim limitations of, "*said access level satisfies the criterion,*" is addressed. The Applicants respectfully assert that claim 1 recites this limitation in two places. Applicants respectfully assert that because the Office Action fails to address each limitation in claim 1 that the Office Action fails to present a prima facie case of unpatentablity. Therefore, the Applicants respectfully request the claim 1 be allowed.

Although the Applicants need not present any further arguments, as the foregoing clearly establishes the patentabilty of claim 1, the Applicants elect to present the further additional arguments.

On pages 3-4 of the final Office Action, remarks were made with respect to "an access level that satisfies a criterion." Applicants respectfully assert that those remarks in no way make up for the defect in the rejection to claim 1 discussed above. As best as the Applicants can understand the remarks, the Office Action appears to allege that every method in a software

-14-

application should have some access level. From there, the Office Action appears to conclude that it would have been obvious to identify the access level of a method, whether a method is synthetic or whether a method calls another method (final Office Action, p. 4, lines 6-7). As an initial matter, Applicants do not concede that it would be obvious to make such determinations regarding a method.

Moreover, Applicants respectfully assert that claim 1 recites, "*modifying said method for a particular purpose only if said method calls another method and said access level satisfies the criterion,*" as claimed. Applicants respectfully assert that they have claimed more than merely *identifying* something about a method. Rather, they have claimed, "*modifying said method for a particular purpose,*" only if "*said method calls another method and said access level satisfies the criterion.*" Applicants respectfully assert that a statement in the Office Action that alleges *identifying* access levels, etc. would be obvious in no way even alleges that the claim limitations would be obvious.

As Applicants have noted above, the solution recited in claim 1 provides an efficient way to determine which methods should be modified, without modifying too many methods. Simply having the ability to identify various aspects about methods in no way teaches or remotely suggests this.

For the foregoing additional reasons, Applicants respectfully assert that claim 1 is patentable.

Next, even if *Berkley,* for the sake of argument, were to be modified to arrive at, "*modifying said method for a particular purpose only if said method calls another method,*" as the Office Action appears to suggest on pages 7-8, *Berkley* would be modifying too many methods. This would be wasteful and inefficient.

Applicants have noted that the solution recited in claim 1 does not necessarily modify all methods that call another method. Claim 1 recites, "*modifying said method for a particular purpose only if said method calls another method and said access level satisfies the criterion.*" Applicants respectfully assert that it is they who have invented the foregoing. Applicants respectfully assert that any assertion that it would be obvious to modify *Berkley* to arrive at the

-15-

claimed invention is pure hindsight reasoning, impermissibly using the Applicants' invention as a blueprint.

Next, *Berkley* by itself does not disclose the claim limitations, *modifying said method for a particular purpose only if said method calls another method and said access level satisfies the criterion,*" as claimed. The Office Action does not appear to contest this point.

Moreover, there is no reason to modify *Berkley* to arrive at these claim limitations. *Berkley* discloses monitoring classes in an object-oriented environment. Applicants respectfully assert that *Berkley* cannot be modified to trace on a method by method basis without **destroying Berkley as a reference**. Stated another way, *Berkley* cannot be modified to trace on a method by method basis without **changing the fundamental principle of operation of Berkley**. Therefore, at the time of Applicants' invention, it would not have been obvious to one of ordinary skill in the art to modify *Berkley* to arrive at the claimed invention.

Berkley discloses monitoring classes in an object-oriented environment. A class may include one or more objects, and the class defines how an object is implemented. Objects are instances of classes (col. 4, lines 61-62). Each object may include one or more methods, as shown in Figure 1. Therefore, a class may contain one or more objects, which each may contain one or more methods. As shown in Figure 1, a method may or may not call another method. For example, in Object 1, which is part of a class, Method A calls Method B. However, Method D does not call any other method.

Berkley discloses monitoring methods by class as follows. If a user wishes to monitor a particular method within a program, the user can change configuration settings associated with the program to specify or activate the class of the method that the user wishes to monitor. "[C]onfiguration settings 300 received as part of a program's execution code are modified at runtime to add a setting to specify (as one example) tracing for a desired class of the executable 310, thereby producing new configuration settings 320" (*Berkley* Figure 5 and col. 6, lines 50-63). *Berkley* further explains that:

> the new configuration settings 330 are then employed with the existing application executable 340 to run the application executable 350. The object

-16-

runtime will query the configuration settings and when a trace is active for a given class, will dynamically insert the trace class into the inheritance hierarchy for that class… Runtime will then see the trace class within the hierarchy and setup for a trace 360 (Berkley col. 6, line 64 – col. 7, line 4).

For example, for a class named 'class 1' that is specified in the configuration settings, the trace is set up through "redirection stubs [that] are created [to] trace entry and exit methods around each target method within class 1" (*Berkley* col. 7, lines 61-63). A trace is active for a given class when the user specifies the class in the configuration settings. The example for activating classes in column 7, lines 20-26 of *Berkley* shows:

> configuration settings used to set up a desired tracing environment. A configuration variable is initialized with the name(s) of the class(es) whose methods should be traced:
> [TraceOptions]
>   ClassTrace = Class1, Class2, Class3, Class4

In the above example, Class1, Class2, Class3, and Class4 are set active by specifying these classes in the ClassTrace list. **All of the methods** in these classes will be traced. Using Figure 1 of *Berkley* as an example, if Object 1 was contained within Class 1, which is specified in the ClassTrace list above, all of the methods contained within Object 1 would be modified for tracing, even Method D which does not call any other method. Since all of the methods are traced, Applicants respectfully asset that *Berkley* cannot be modified to arrive at the claimed invention without destroying *Berkley* as a reference.

Stated another way to attempt to modify *Berkley* to arrive at the claimed invention would fundamentally alter the principle of operation of *Berkley*. One of ordinary skill in the art would not have had a reason to modify *Berkley* given this need to fundamentally alter the principle of operation.

As the foregoing clearly states, *Berkley* discloses that tracing performed by activating **classes**. *Berkley* does not disclose a technique for a user to specify a particular method to trace. Therefore, *Berkley* cannot be modified to arrive at, "*modifying said method for a particular*

purpose only if said method calls another method and said access level satisfies the criterion," as claimed.

   Webster is concerned with providing a system that allows a user to specify methods to trace, and to specify what information about those methods the user would like. However, the system that *Webster* has taught does not disclose the claim limitations. Moreover, there is simply no reason to modify *Webster's* system to arrive at the claimed invention or to combine *Webster's* teachings with another reference to arrive at the claimed invention.

   Webster discloses that a file is provided that *lists* methods to trace. Applicants first note that *Webster* teaches that a user provides the file. Therefore, in *Webster* the **user has control** over specifically what methods to trace. The claim limitations are for **automatically** determining whether to modify said method. Automating the process of producing the list of methods and classes destroys *Webster* as a reference. Stated another way, it would fundamentally change the principle of operation of *Webster* to automate the process of allowing providing the list. Note that the user would no longer have control to specify what methods to trace, if this change were made to *Webster*.

   Furthermore, *Webster* provides no teaching or suggestion to put the following into the list of methods to trace: "whether said method calls another method and if said method has an access level that satisfies a criterion." *Webster* does not in any way teach or suggest either of these, much less the combination. *Webster* does not teach or suggest that a method that calls another method should be added to the list of methods to trace. *Webster* does not teach or suggest that has an access level that satisfies a criterion should be added to the list of methods to trace.

   Moreover, since neither *Webster* nor *Berkley* disclose, "automatically determining whether to modify said method, said step of automatically determining whether to modify said method includes automatically determining whether said method calls another method and if said method has an access level that satisfies a criterion," the combination does not disclose these limitations.

-18-

Grove discloses that it is difficult to accurately construct call graphs. Applicants refer to "*A Framework for Call Graph Construction Algorithms* (David Grove and Craig Chambers) (herein "*Grove and Chambers*"), which states on page 686 that in object oriented languages and languages with function values, the target of a call cannot always be precisely determined solely by an examination of the source code of the call site. *Grove and Chambers* states that an inter-procedural analysis may need to be performed, which actually requires that a call graph be built prior to the analysis being performed. Thus, *Grove and Chambers* state that this circularity needs to be resolved for "higher level languages" (which *Grove and Chambers* define as object oriented languages and functional languages.

Applicants have recited a solution in claim 1 that does not require that a call graph be constructed. Rather, the Applicants have discovered that a much simpler solution is possible.

Moreover, Applicants respectfully assert that one of ordinary skill in the art having *Grove* (or *Grove and Chambers*) in front of them would not find it obvious or have any reason to modify *Berkley* in view of *Webster* to arrive at the claim limitations. Applicants respectfully assert that all *Grove* discloses or suggests is a technique for solving the complexities of determining call graphs. Even if *Grove* was used to produce a call graph, which is then presented to the user in *Webster*, the Applicants' claimed invention still does not present itself to the person of ordinary skill in the art. Note that even if, for the sake of argument, the user were to add only methods that are at the top of the call graph to the list of methods to analyze in *Webster*, **the claim limitations are not met**.

Applicants respectfully assert that there is nothing in *Grove*, even in view of *Berkley* in view of *Webster* that discloses *modifying said method for a particular purpose only if said method calls another method and said access level satisfies the criterion.* For example, simply presenting a call graph in no way supplies what is missing in *Berkley* in view of *Webster*. As noted, *Berkley* discloses monitoring classes in an object-oriented environment. Merely teaching how to construct a call graph does not lead one to the solution of what methods are modified in claim 1. As noted, *Webster* discloses allowing a user to provide a list of methods to trace. Merely teaching how to construct a call graph does not lead one to the solution of **that a method should be added to the list** *only if said method calls another method and said access level*

-19-

satisfies the criterion.

Applicants note that they have taught a fundamentally different technique than any combination of *Berkley*, *Webster*, and *Grove*. What Applicants have recited in claim 1 does not to use brute force to determine a call graph (which may be inaccurate anyway). Instead, claim 1 recites an elegant solution that modifies a method for a particular purpose *only if said method calls another method and said access level satisfies the criterion.*

For all of the foregoing reasons, Berkley in view of Webster in further view of Grove fails to disclose, "*modifying said method for a particular purpose only if said method calls another method and said access level satisfies the criterion,*" as claimed.

ii) Claim 13 is patentable under 35 U.S.C. §103(a) over *Berkley* in view of *Webster* and *Grove*

Claim 13 recites:

> A process for monitoring, comprising:
> automatically determining which methods of a set of methods call one or more other methods and are synthetic; and
> using a first tracing mechanism for said *methods that call one or more other methods and are not synthetic* without using said first tracing mechanism for methods that do not call one or more other methods or are synthetic.

Applicants respectfully assert that *Berkley* in view of *Webster* in further view of *Grove* fails to disclose, "*using a first tracing mechanism for said methods that call one or more other methods and are not synthetic without using said first tracing mechanism for methods that do not call one or more other methods or are synthetic,*" as claimed.

As stated in paragraph 37 of the instant specification, "a synthetic method is a method that does not appear in the source code. The rejection to claim 13 seems to hinge on an argument that *Webster* discloses a selective trace coupled with an argument that it is possible to identify

<div align="center">-20-</div>

what methods are synthetic and what methods call other methods.

Applicants respectfully assert that it would not be obvious to use a first tracing mechanism for methods that are both synthetic **and** call one or more other methods, even if synthetic methods are known in JAVA programming techniques. The Office Action appears to assert that synthetic methods are widely used and known (final Office Action, page 13, lines 2-4). However, simply knowing of the existence of synthetic methods does not in any way teach or suggest *using(or not using) a first tracing mechanism based on whether a method is synthetic.*

Next, even if a first tracing mechanism were used for all methods that are not synthetic, the claim limitations would not be met. This would use the first tracing mechanism for methods regardless of whether they call one or more methods. As Applicants have noted above, Applicants have solved a problem of tracing **too many** methods. Moreover, Applicants have created a solution that is both elegant and simple to implement.

On the other hand, if a first tracing mechanism were used for all methods that called other methods, the claim limitations would not be met. This would use the first tracing mechanism regardless of whether a method is synthetic. As Applicants have noted above, Applicants have solved a problem of tracing **too many** methods. Moreover, Applicants have created a solution that is both elegant and simple to implement.

For the foregoing reasons, Applicants respectfully assert that claim 13 is patentable.

While the Applicants need not present any further arguments, the Applicants will discuss the references in more detail. *Berkley* by itself does not disclose these claim limitations pertaining to tracing **based on whether a method is synthetic**. Moreover, claim 13 has the additional limitation of *methods that call one or more other methods*. Thus, first tracing mechanism is used for said *methods that call one or more other methods **and** are not synthetic.*

Likewise, *Webster* in further view of *Grove* do not teach these claim limitations. As noted above, *Webster* discloses that a user can provide file that *lists* methods to trace. Applicants first note that *Webster* teaches that a user provides the file. Therefore, in *Webster* the user determines whether to trace the method. The claim limitations are for **automatically** determining whether to modify said method. Moreover, automating the process of producing the

-21-

list of methods and classes destroys *Webster* as a reference. Stated another way, it would fundamentally change the principle of operation of *Webster* to automate the process of the user providing the list.

Furthermore, ***Webster provides no teaching or suggestion to put the following into the list*** of methods to trace "methods that call one or more other methods and are not synthetic." *Webster* does not in any way teach or suggest either of these, much less the combination. *Webster* does not teach or suggest that a method that calls another method should be added to the list of methods to trace. *Webster* does not teach or suggest that methods that are not synthetic should be traced.

Moreover, *Grove*'s discussion on constructing call graphs does not remedy the foregoing deficiencies. For the foregoing reasons, Applicants respectfully assert that claim 13 is patentable.

Moreover, there is no reason to modify *Berkley* to arrive at these claim limitations. *Berkley* discloses monitoring classes in an object-oriented environment. Applicants respectfully asset that *Berkley* cannot be modified to trace on a method by method basis without **destroying** *Berkley* **as a reference**. Stated another way, *Berkley* cannot be modified to trace on a method by method basis without **changing the fundamental principle of operation of** *Berkley*. Therefore, at the time of Applicants' invention, it would not have been obvious to one of ordinary skill in the art to modify *Berkley* to arrive at the claimed invention.

Applicants note that they have taught a fundamentally different technique than any combination of *Berkley*, *Webster*, and *Grove*. Applicant's invention recited in claim 13 is not to use brute force to determine a call graph (which may be inaccurate anyway). Instead, claim 13 recites an elegant solution of "*using a first tracing mechanism for said methods that call one or more other methods and are not synthetic without using said first tracing mechanism for methods that do not call one or more other methods or are synthetic.*"

-22-

For all of the foregoing reasons, Berkley in view of Webster in further view of Grove fails to disclose, "*using a first tracing mechanism for said methods that call one or more other methods and are not synthetic without using said first tracing mechanism for methods that do not call one or more other methods or are synthetic*," as claimed.

iii) Claim 40 is patentable under 35 U.S.C. §103(a) over *Berkley* in view of *Webster* and *Grove*

Applicants respectfully assert that *Berkley* in view of *Webster* in further view of *Grove* fails to disclose, "*tracing said method for a particular purpose only if said method calls one or more different methods and can be called by a sufficient scope of one or more other methods*," as recited in claim 40.

Berkley by itself does not disclose these claim limitations. Claim 40 has the limitation of *methods that call one or more other methods*. Moreover, claim 40 has the additional limitation of *said method can be called by a sufficient scope of one or more other methods*. Note that this results in a far different set of methods being traced than tracing **both** methods that *call one or more other methods* and methods that *can be called by a sufficient scope of one or more other methods*. Applicants have taught a way to **limit** the number of methods traced, while not resorting to complex techniques such as building call graphs.

Applicants respectfully assert that *Berkley* at col. 2, lines 41-43 in no way discloses these claim limitations. *Berkley* merely mentions a "means for dynamically creating a redirection stub to insert the function for the class if the function is active for the class." This in no way discloses tracing based on the combination of whether a method calls another method and can be called by a sufficient scope of one or more methods. Applicants respectfully note that the Office Action concedes that *Berkley* does not disclose, "determining whether said method calls one or more different methods and can be called by a sufficient scope of one or more other methods." If *Berkley* fails to make such a determination, then *Berkley* cannot teach tracing based on the claimed combination.

-23-

Moreover, it would not be obvious to trace methods that both call one or more other methods and can be called by a sufficient scope of one or more other methods, even if calling scope is known in JAVA programming techniques. First of all, even all if methods of a certain scope were to be traced, the claim limitations would not be met. Tracing all methods of sufficient scope would trace those that **do not** call one or more methods. Thus, **too many** methods would be traced.

Furthermore, *Webster* in further view of *Grove* do not remedy these deficiencies. As noted above, *Webster* discloses that a user can provide a file that *lists* methods to trace. *Webster* does not teach or suggest that a method that calls another method should be added to the list of methods to trace. *Webster* does not teach or suggest that methods that method can be called by a sufficient scope of one or more other methods should be traced.

Moreover, *Grove*'s discussion on constructing call graphs does not remedy the foregoing deficiencies.

Moreover, there is no reason to modify *Berkley* to arrive at these claim limitations. *Berkley* discloses monitoring classes in an object-oriented environment. Applicants respectfully asset that *Berkley* cannot be modified to trace on a method by method basis without **destroying** ***Berkley* as a reference**. Stated another way, *Berkley* cannot be modified to trace on a method by method basis without **changing the fundamental principle of operation of *Berkley***. Therefore, at the time of Applicants' invention, it would not have been obvious to one of ordinary skill in the art to modify *Berkley* to arrive at the claimed invention.

Applicants note that they have taught a fundamentally different technique than any combination of *Berkley*, *Webster*, and *Grove*. Applicant's invention recited in claim 40 is not to use brute force to determine a call graph (which may be inaccurate anyway). Instead, claim 40 recites an elegant solution of *"tracing said method for a particular purpose only if said method calls one or more different methods and can be called by a sufficient scope of one or more other methods."*

-24-

For all of the foregoing reasons, Berkley in view of Webster in further view of Grove fails to disclose, "*tracing said method for a particular purpose only if said method calls one or more different methods and can be called by a sufficient scope of one or more other methods,*" as claimed.

B) Rejection of Claims 9, 11, 19, 21-22, 28-30, 32, 38 and 45-46 Under 35 U.S.C. §103(a) as being unpatentable over *Berkley in view of Webster, Grove, and Berry*

Applicants respectfully assert that the rejection of Claims 9, 11, 19, 21-22, 28-30, 32, 38 and 45-46 under 35 U.S.C. §103(a) as being unpatentable over *Berkley* in view of *Webster* and *Grove*, and in further view of U.S. Patent No. 6,662,359 to *Berry et al.* (hereinafter "*Berry*") is in error. The rejection is respectfully traversed for the following reasons.

i) Claim 22 is patentable under 35 U.S.C. §103(a) over *Berkley* in view of *Webster, Grove, and Berry*

Claim 22 recites:

> One or more processor readable storage devices having processor readable code embodied on said processor readable storage devices, said processor readable code for programming one or more processors to perform a process comprising:
>> automatically determining which methods of a set of methods to modify, said step of determining includes automatically determining which methods call one or more other methods and have an access level of either public or package in the JAVA programming language; and
>> modifying for a particular purpose only those methods that call one or more other methods and have an access level of either public or package in the JAVA programming language.

Berkley in view of *Webster,* in further view of *Grove,* and in still in further view of *Berry* fails to disclose, "*modifying for a particular purpose only those methods that call one or more other methods and have an access level of either public or package in the JAVA programming*

-25-

language," as claimed.

Berkley by itself does not disclose these claim limitations pertaining to *modifying for a particular purpose only those methods that... have an access level of either public or package in the JAVA programming language,*" as claimed. Moreover, claim 22 has the additional limitation of *methods that call one or more other methods.* Thus, *modifying for a particular purpose* is used for said *methods that call one or more other methods **and** have an access level of either public or package in the JAVA programming language.*

Moreover, it would not be obvious to "*modify for a particular purpose*" those methods that both *have an access level of either public or package in the JAVA programming language **and** call one or more other methods,* even access levels are known in JAVA programming techniques. First of all, even if methods having a certain access level were to be traced, the claim limitations would not be met. Tracing all methods having an access level of *public or package in the JAVA programming language* would *modify for a particular purpose* those that **do not** call one or more methods. Thus, too many methods would be *modified for the particular purpose.* Applicants have solved a problem of modifying **too many** methods. Moreover, Applicants have created a solution that is both elegant and simple to implement.

Likewise, *Webster* in further view of *Grove* and *Berry* do not teach these claim limitations. *Webster* provides no teaching or suggestion to put the following into the list of methods, "*methods that call one or more other methods and have an access level of either public or package in the JAVA programming language.*" *Webster* does not in any way teach or suggest either of these, much less the combination. *Webster* does not teach or suggest that a method that calls another method should be added to the list of methods. *Webster* does not teach or suggest that methods that *have an access level of either public or package in the JAVA programming language* should be added to the list.

Moreover, Applicants respectfully assert that one of ordinary skill in the art having *Grove* in front of them would not find it obvious or have any reason to modify *Berkley* in view of *Webster* to arrive at the claim limitations. Applicants respectfully assert that all *Grove* discloses or suggests is a technique for solving the complexities of determining call graphs. Even if *Grove*

-26-

was used to produce a call graph, which is then presented to the user in *Webster*, the Applicants' claimed invention still does not present itself to the person of ordinary skill in the art. Note that even if, for the sake of argument, the user were to add only methods that are at the top of the call graph to the list of methods to analyze in *Webster*, ***the claim limitations are not met***.

It does not appear to the Applicants that the Office Action alleges *Berry* discloses these limitations. *Berry* has apparently been used as allegedly disclosing limitations with respect to inserting hooks into JAVA classes (see Office Action, page 22). Therefore, *Berkley* in view of *Webster* in further view of *Berry* does not disclose the claim limitations.

Moreover, there is no reason to modify *Berkley* to arrive at these claim limitations. *Berkley* discloses monitoring classes in an object-oriented environment. Applicants respectfully asset that *Berkley* cannot be modified to trace on a method by method basis without **destroying** ***Berkley* as a reference**. Stated another way, *Berkley* cannot be modified to trace on a method by method basis without **changing the fundamental principle of operation of *Berkley***. Therefore, at the time of Applicants' invention, it would not have been obvious to one of ordinary skill in the art to modify *Berkley* to arrive at the claimed invention.

Applicants note that they have taught a fundamentally different technique than any combination of *Berkley*, *Webster*, *Grove*, and *Berry*. Applicant's invention recited in claim 22 is not to use brute force to determine a call graph (which may be inaccurate anyway). Instead, claim 22 recites an elegant solution of "*modifying for a particular purpose only those methods that call one or more other methods and have an access level of either public or package in the JAVA programming language.*"

For all of the foregoing reasons, *Berkley* in view of *Webster* in further view of *Grove*, in still further view of *Berry* fails to disclose, "*modifying for a particular purpose only those methods that call one or more other methods and have an access level of either public or package in the JAVA programming language,*" as claimed.

-27-

ii) Claim 33 is patentable under 35 U.S.C. §103(a) over *Berkley* in view of *Webster, Grove, and Berry*

Berkley in view of *Webster* in further view of *Grove* in still further view of *Berry* fails to disclose, "*tracing said method for a particular purpose only if said method calls another method and said access level satisfies the criterion*," as recited in claim 33.

The Office Action alleges that limitations of claim 33 were discussed in the rejection of claims 13-15 and 19. Applicants respectfully assert that the rejection of claims 13-15 and 19 does not in fact address all limitations of claim 33.

Claim 13 has claim limitations of, "*using a first tracing mechanism for said methods that call one or more other methods and are not synthetic without using said first tracing mechanism for methods that do not call one or more other methods or are synthetic.*"

Applicants respectfully assert that, "*said access level satisfies the criterion*" is not equivalent to "*methods ... are synthetic.*"

As stated in the Specification:

> In one embodiment, a method is complex if it meets three criteria: (1) the method has an access level of public or package; (2) the method is non-synthetic and (3) the method calls at least one other method. Methods that do not satisfy all three criteria are classified as simple methods. In other embodiments, a method can be classified as complex if it satisfies two of the above criteria, or other similar criteria (p. 12, lines 17-23).

One example of an access level satisfying a criterion is for the method to have an access level of public or package. However, as noted other similar criteria could be used. Note that the access level of the method is quite different from whether the method is synthetic. As noted in the Specification regarding the access level:

> Java provides for four levels of access control for methods: public, private, protected, and package. Private is the most restrictive access level. A private method can only be accessed by methods in the same class. A protected method can be accessed by other methods in the same class, sub classes and classes in the same package. A public method can be accessed by any class of any parentage in any package. The package access level is the default and allows a method to be

-28-

accessed by classes in the same package, regardless of their parentage" (p. 12, line 24 – p. 13, line 5).

On the other hand, a method may be determined to be synthetic if it is "a method that does not appear in the source code. For example, during compilation, the compiler may add one or more methods. Typically, the compiler explicitly makes it easy to see that methods are or are not synthetic. Java compilers flag these methods with the 'Synthetic' attribute" (Specification p. 13, lines 6-11).

Thus, Applicants respectfully assert that when the rejection addressed the claim limitations of *"using a first tracing mechanism for said methods that call one or more other methods and are not synthetic without using said first tracing mechanism for methods that do not call one or more other methods or **are synthetic**,* the Office Action in no way addressed all claim limitations of claim 33.

Next, Applicants respectfully note that claims 14-15 have been cancelled, without prejudice.

Finally, claim 19 recites, "said using a first tracing mechanism includes adding and using timers for said methods." Applicants respectfully assert that the Office Action did not address claim limitations pertaining to, *"tracing said method for a particular purpose only if said method calls another method and **said access level satisfies the criterio**n,"* when addressing claim 19 (please see final Office Action at p. 19-20).

Because the Office Action has failed to address each and every limitation in the rejection of claim 33, Applicants respectfully assert that claim 33 is allowable.

Although Applicants need not present any further arguments, Applicants will briefly discuss the references. Applicants have already addressed these claim limitations with respect to *Berkley* in view of *Webster* and in further view of *Grove* their response to claim 1. *Berry* has apparently been used as allegedly disclosing limitations with respect to inserting hooks into JAVA classes (see rejection of claim 46). However, Applicants do not believe that *Berry* is alleged to disclose, "tracing said method for a particular purpose only if said method calls another method and said access level satisfies the criterion," as claimed.

-29-

Therefore, Applicants respectfully assert that claim 33 is patentable over *Berkley* in view of *Webster* in further view of *Grove* in still further view of *Berry*.

CONCLUSION

Based on the above, it is respectfully submitted that claims 1-2, 5-13, 17-23, 28-30, 32-35, 37-42, 44-47, 51 and 52 are patentable over the cited references, and it is respectfully requested that the rejection of claims 1-2, 6-13, 18-22, 28-30, 32-35, 38, 40, 42, 44-46, and 52 and the objection of claims 5, 17, 23, 37, 41 and 51 be withdrawn.

The Commissioner is authorized to charge any underpayment or credit any overpayment to Deposit Account No. 501826 for any matter in connection with this Appeal Brief, including any fee for extension of time, which may be required.

Respectfully submitted,

Date: __August 3, 2012__   By: __/RonaldMPomerenke#43009/__
                                          Ronald M. Pomerenke
                                          Reg. No. 43,009

VIERRA MAGEN MARCUS & DENIRO LLP
575 Market Street, Suite 2500
San Francisco, CA 94105-4206
Telephone: (415) 369-9660
Facsimile: (415) 369-9665

-30-

V. CLAIMS APPENDIX *(37 C.F.R. §41.37(c)(1)(v))*

1.   (previously presented) A process for monitoring, comprising:

accessing a method;

automatically determining whether to modify said method, said step of automatically determining whether to modify said method includes automatically determining whether said method calls another method and whether said method has an access level that satisfies a criterion; and

modifying said method for a particular purpose only if said method calls another method and said access level satisfies said criterion.

2.   (previously presented) The process according to claim 1, wherein:

said determining whether to modify said method includes automatically determining whether said method is a synthetic method; and

said modifying includes modifying said method only if said method is not a synthetic method, said access level satisfies said criterion and said method calls another method.

3-4.   (cancelled)

5.   (previously presented) The process according to claim 1, wherein:

said determining whether to modify said method includes automatically determining whether said method is a synthetic method and has an access level of public or package in the JAVA programming language; and

said modifying includes modifying said method only if said method is not a synthetic method, has said access level of public or package, and said method calls another method.

6.   (previously presented) The process according to claim 1, wherein:

said determining whether said method has an access level that satisfies a criterion includes automatically determining whether said method can be called by a sufficient scope of

-31-

one or more other methods; and

said modifying said method includes modifying said method only if said method can be called by said sufficient scope of one or more other methods and said method calls another method.

7.   (previously presented) The process according to claim 1, wherein:
said modifying includes modifying object code.

8.   (previously presented) The process according to claim 1, wherein:
said modifying includes adding a tracer for said method.

9.   (previously presented) The process according to claim 1, wherein:
said modifying includes adding a timer for said method.

10.   (previously presented) The process according to claim 1, wherein:
said modifying includes adding exit code and start code to existing object code.

11.   (previously presented) The process according to claim 10, wherein:
said start code starts a tracing process;

said exit code stops said tracing process;

said exit code is positioned to be executed subsequent to original object code;

said adding exit code includes adding an instruction to jump to said exit code from said original object code;

said adding exit code includes adding an entry in an exceptions table; and

said adding an entry in said exceptions table includes adding a new entry into said exceptions table for said method, said new entry indicates a range of indices corresponding to said original object code, said new entry includes a reference to said exit code and said new entry indicates that said new entry pertains to all types of exceptions.

-32-

12.   (previously presented) The process according to claim 1, wherein:

said particular purpose is to add a first tracer.

13.   (previously presented) A process for monitoring, comprising:

automatically determining which methods of a set of methods call one or more other methods and are synthetic; and

using a first tracing mechanism for said methods that call one or more other methods and are not synthetic without using said first tracing mechanism for methods that do not call one or more other methods or are synthetic.

14-16. (cancelled)

17.   (previously presented) The process according to claim 13, wherein:

said automatically determining includes automatically determining whether said methods have an access level of public or package in the JAVA programming language; and

said using includes using said first tracing mechanism only if said methods are not synthetic methods, have said access level of public or package in the JAVA programming language, and said methods call one or more other methods.

18.   (previously presented) The process according to claim 13, wherein:

said determining includes automatically determining whether said methods can be called by a sufficient scope of one or more other methods; and

said using includes using said first tracing mechanism only if said methods can be called by said sufficient scope of one or more other methods, said methods are not synthetic, and said methods call one or more other methods.

19.   (previously presented) The process according to claim 13, wherein:

said using a first tracing mechanism includes adding and using timers for said methods.

20.   (previously presented) The process according to claim 13, wherein:

said using a first tracing mechanism includes modifying existing object code to add said first tracing mechanism.

21.   (previously presented) The process according to claim 20, wherein:

said first tracing mechanism includes timers for said methods.

22.   (previously presented) One or more processor readable storage devices having processor readable code embodied on said processor readable storage devices, said processor readable code for programming one or more processors to perform a process comprising:

automatically determining which methods of a set of methods to modify, said step of determining includes automatically determining which methods call one or more other methods and have an access level of either public or package in the JAVA programming language; and

modifying for a particular purpose only those methods that call one or more other methods and have an access level of either public or package in the JAVA programming language.

23.   (previously presented) The one or more processor readable storage devices according to claim 22, wherein:

said automatically determining includes automatically determining whether said methods are not synthetic methods; and

said modifying includes modifying said methods only if said methods are determined to not be synthetic methods, said methods have an access level of either public or package in the JAVA programming language and said methods call one or more other methods.

24-27. (cancelled)

28.   (previously presented) The one or more processor readable storage devices according to claim 22, wherein:

-34-

said modifying includes modifying existing object code.

29. (previously presented) The one or more processor readable storage devices according to claim 22, wherein:

said modifying includes adding tracers.

30. (previously presented) The one or more processor readable storage devices according to claim 22, wherein:

said modifying includes adding timers.

31. (cancelled)

32. (previously presented) The one or more processor readable storage devices according to claim 22, wherein:

said start code starts a tracing process;

said exit code stops said tracing process;

said exit code is positioned to be executed subsequent to original object code;

said step of adding exit code includes adding an instruction to jump to said exit code from said original object code;

said adding exit code includes adding an entry in an exceptions table; and

said adding an entry in said exceptions table includes adding a new entry into said exceptions table for said method, said new entry indicates a range of indices corresponding to said original object code, said new entry includes a reference to said exit code and said new entry indicates that said new entry pertains to all types of exceptions.

33. (previously presented) One or more processor readable storage devices having processor readable code embodied on said processor readable storage devices, said processor readable code for programming one or more processors to perform a process comprising:

automatically determining whether to trace a method, said step of determining includes

automatically determining whether said method calls another method and if said method has an access level that satisfies a criterion; and

said method for a particular purpose only if said method calls another method and said access level satisfies the criterion.

34. (previously presented)   The one or more processor readable storage devices according to claim 33, wherein:

said determining   includes automatically determining whether or not said method is flagged by a compiler as being synthetic; and

said tracing includes tracing said method only if said method is not flagged by said compiler as being synthetic and said method calls another method.

35. (previously presented)   The one or more processor readable storage devices according to claim 33, wherein:

said automatically determining includes automatically determining whether said method has an access level of public or package in the JAVA programming language, an access level of public indicates that a method can be called by a method in a class of any parentage, an access level of package indicates that a method can be called by methods in classes in the same package regardless of parentage; and

said tracing includes tracing said method only if said method is determined to have said access level of public or package and said method calls another method.

36. (cancelled)

37. (previously presented)   The one or more processor readable storage devices according to claim 33, wherein:

said automatically determining includes automatically determining whether said method is not a synthetic method and has an access level of public or package, said access level is one of a plurality of access levels in a JAVA programming language; and

-36-

said tracing includes tracing said method only if said method is determined to not be a synthetic method, have said access level of public or package, and said method calls another method.

38.   (previously presented)   The one or more processor readable storage devices according to claim 33, wherein:

said tracing includes timing said method.

39.   (previously presented) An apparatus capable of monitoring, comprising:

means for automatically determining whether a method calls another method;

means for automatically determining whether said method can be called by a sufficient scope of one or more other methods;

means for automatically determining whether said method is not a synthetic method; and

means for tracing said method for a particular purpose only if said method calls another method, said method can be called by a sufficient scope of one or more other methods, and said method is not a synthetic method.

40.   (previously presented) An apparatus capable of monitoring, comprising:

a storage device; and

one or more processors in communication with said storage device, said one or more processors perform a process comprising:

accessing a method,

determining whether said method calls one or more different methods and can be called by a sufficient scope of one or more other methods, and

tracing said method for a particular purpose only if said method calls one or more different methods and can be called by a sufficient scope of one or more other methods.

41.   (previously presented) The apparatus according to claim 40, wherein:

said determining includes determining whether said method is not a synthetic method;

-37-

and

said tracing includes tracing said method only if said method is determined to not be a synthetic method and said method calls one or more different methods.

42.   (previously presented) The apparatus according to claim 40, wherein:

said determining includes determining whether said method has an access level of public or package in the JAVA programming language; and

said tracing includes tracing said method only if said method is determined to have said access level of public or package and said method calls one or more different methods.

43.   (cancelled)

44.   (previously presented) The apparatus according to claim 40, wherein:

said process further includes modifying existing object code for said method in order to add a first tracing mechanism.

45.   (previously presented) The apparatus according to claim 44, wherein:

said first tracing mechanism includes a timer.

46.   (previously presented) The apparatus according to claim 40, wherein:

said tracing includes timing said method.

47.   (previously presented) A process for monitoring, comprising:

accessing a method;

automatically determining whether said method is complex, said step of automatically determining includes automatically determining that said method is complex if said method satisfies the following criteria:

said method calls another method;

said method has an access level of public or package in the JAVA programming

-38-

language; and

said method is not flagged by a compiler as being synthetic; and

adding a tracer to said method only if said method is automatically determined to be complex.

48-50. (cancelled)

51. (previously presented) The process according to claim 5, wherein:

said modifying includes adding a tracer for said method.

52. (previously presented) The apparatus according to claim 40, wherein:

said determining includes determining whether said method is not a synthetic method and whether said method has an access level of public or package in the JAVA programming language; and

said tracing includes tracing said method only if said method is determined to not be a synthetic method, said method is determined to have an access level of public or package, and said method calls one or more different methods.

53-59. (cancelled)